HANGMAN GAME TEST

Xingrong Zong

Email: xz222bb@student.lnu.se

Gitlab repo: https://gitlab.lnu.se/1dv600/student/xz222bb/assignment-3

AAA

02.04.2020

Contents

1. Test Plan

1.1 Objectives of the Testing

The test's goal is to check that the functionalities are in compliance with these requirements. Manual tests will be conducted to verify the system's functionalities from the user side. Unit tests will be conducted to verify the system's functionalities from the developer side.

1.2 What to Test and How?

There are in total three Use Cases: Start Game, Play Game and Quit Game. For this iteration, there are two use cases be chosen to be tested by writing and running dynamic manual test cases. Moreover, some methods from HangmanManager class: playerWon(), playerLost(), endGame(), getStatus() and hangman() will be tested using automated unit tests.

1.3 Testing Approach

Testing Execution Process

Each method has an associated function test that feeds the function a set of inputs then compares the actual outputs with the expected ones. The test fails if one or more of the actual outputs differs significantly from the expected ones.

Functional Testing of the System

As already mentioned, the testing will not be extensive due to the modesty of the software in question, and in the project's current state.

Purpose: Functional testing will be conducted to verify features listed for the application that will be useful for regression testing in the future.

Method: The tests will be performed according to functional scripts, which are stored in a test class. These test scripts will be updated if a new testing scenario presents itself.

1.4 Time Log

Task	Estimated Time	Actual Time	Time Difference
Create a Project Plan	2.5h	3.5h	1h
Update the Project Plan	30min	30min	0
Create a Time Log Table	10min	25min	15min
Update the Time Log table	5min	5min	0
Create a Use Cases Diagram	25min	35min	10min
Create the fully dressed Use Cases	1.5h	2.5h	1h
Create a State Machine Diagram	40min	100h	1h
Create a Class Diagram	25min	55min	30min
Read materials	6h	8h	2h
Update materials	2h	2h	0

Implement the code for the Hangman Game	6h	11h	5h
Update the code	1h	2h	1h
Create Test Plan	1h	1h	0
Create Manual Test	2h	2.5h	0.5h
Create Unit Test	2h	2h	0
Reflection	20min	20min	0

2. Manual Test Cases using the client application

TC1.1: Start Game Successfully

Use case: UC 1 Start Game

Scenario: Start the game successfully

Description: The main scenario of UC1 is tested where a user starts game successfully.

Precondition: The only word option from Vocabulary library is “acquiesce”

Test Steps:

1. Run the project
2. System shows
   “ Welcome to Hangman! Let's start!
      1) Play Game
      2) Quit Game”
3. Type 1 and press enter

Expected Results:

_ _ _ _ _ _ _ _

Wrong guesses:

Enter a letter (to quit type 2):

Actual Results: **Pass**

TC1.2: Quit Game Successfully

Use case: UC 1 Start Game

Scenario: Quit the game successfully

Description: The main scenario of UC1 is tested where a user quits game successfully.

Precondition: The only word option from Vocabulary library is “acquiesce”

Test Steps:

1. Run the project
2. System shows

   “ Welcome to Hangman! Let's start!

   1) Play Game

   2) Quit Game”
3. Type 2 and press enter

Expected Results:

Process finished with exit code 1

Actual Results: **Pass**

TC2.1: Invalid input and promote input again

Use case: UC 2 Play Game

Scenario: The player types in invalid things

Description: The main scenario of UC2 is tested where a user types in invalid answers.

Precondition:

The game is running.

The only word option from Vocabulary library is “acquiesce”

Test Steps with Expected Results:

1. System shows the underlines as the number of letters of word:

_ _ _ _ _ _ _ _ _

Wrong guesses:

Enter a letter (to quit type 2):

2. Type “5” and press enter.
3. System checks, the input is not a letter and shows:

Please enter a letter.

4. Repeat from step 2 until the input is letter:

The word was acquiesce.

Actual Results: **Pass**

TC2.2: Player wins the game without any wrong guess

Use case: UC 2 Play Game

Scenario: The player wins the game without any wrong guess

Description: The main scenario of UC2 is tested where a user types in invalid answers.

Precondition:

The game is running.

The only word option from Vocabulary library is “acquiesce”

Test Steps with Expected Results:

1. System shows the underlines as the number of letters of word:

_ _ _ _ _ _ _ _ _

Wrong guesses:

Enter a letter (to quit type 2):

2. Type “a” and press enter.
3. System checks, the letter is correct and shows on the corresponding position.

a _ _ _ _ _ _ _ _

Wrong guesses:

Enter a letter (to quit type 2):

4. Repeat from step 2 (c, q, u, i, e, s):

a c q u i e s c e

YOU SUCCEED

> The word was acquiesce.

Actual Results: **Pass**

TC2.3: Player loses the game without any correct guess

Use case: UC 2 Play Game

Scenario: The player loses the game without any correct guess

Description: The main scenario of UC2 is tested where a user types in invalid answers.

Precondition: The only word option from Vocabulary library is “acquiesce”

Test Steps with Expected Results:

1. System shows the underlines as the number of letters of word:

_ _ _ _ _ _ _ _ _

Wrong guesses:

Enter a letter (to quit type 2):

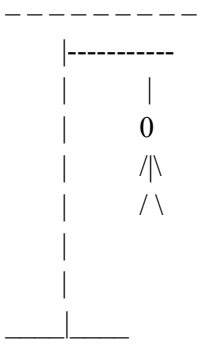
2. Type “b” and press enter.
3. System checks, the letter is correct and shows on the corresponding position.

_ _ _ _ _ _ _ _ _

Wrong guesses: b

Enter a letter (to quit type 2):

4. Repeat from step 2 nine more times with wrong guess:



YOU DIED

The word was acquiesce.

Actual Results: **Pass**

Test Report:

Test	UC1	UC2
TC1.1	1/OK	0
TC1.2	1/OK	0
TC2.1	0	1/OK
TC2.2	0	1/OK
TC2.3	0	1/OK

3. Unit Tests

Test playerWon():

```
@org.junit.jupiter.api.Test
void playerWon() {
    HangmanManager h = new HangmanManager();
    assertFalse(h.playerWon());
    h.setAnswer("you");
    ArrayList<Character> word = new ArrayList<~>();
    word.add('y');
    word.add('o');
    word.add('u');
    h.setCorrectGuess(word);
    assertTrue(h.playerWon());
}
```

Test playerLost():

```
@org.junit.jupiter.api.Test
void playerLost() {
    HangmanManager h = new HangmanManager();
    assertFalse(h.playerLost());
    h.setAnswer("what");
    ArrayList<Character> word = new ArrayList<~>();
    word.add('y');
    word.add('o');
    word.add('u');
    word.add('y');
    word.add('o');
    word.add('u');
    word.add('y');
    word.add('o');
    word.add('u');
    word.add('y');
    word.add('o');
    h.setWrongGuess(word);
    assertTrue(h.playerLost());
}
```

Test endgame():

```
@org.junit.jupiter.api.Test
void endGame() {
    HangmanManager h = new HangmanManager();
    assertFalse(h.endGame());
    h.setAnswer("you");
    ArrayList<Character> word = new ArrayList<~>();
    word.add('y');
    word.add('o');
    word.add('_');
    h.setCorrectGuess(word);
    assertFalse(h.endGame());

    word.clear();
    word.add('y');
    word.add('o');
    word.add('u');
    assertTrue(h.endGame());
}
```

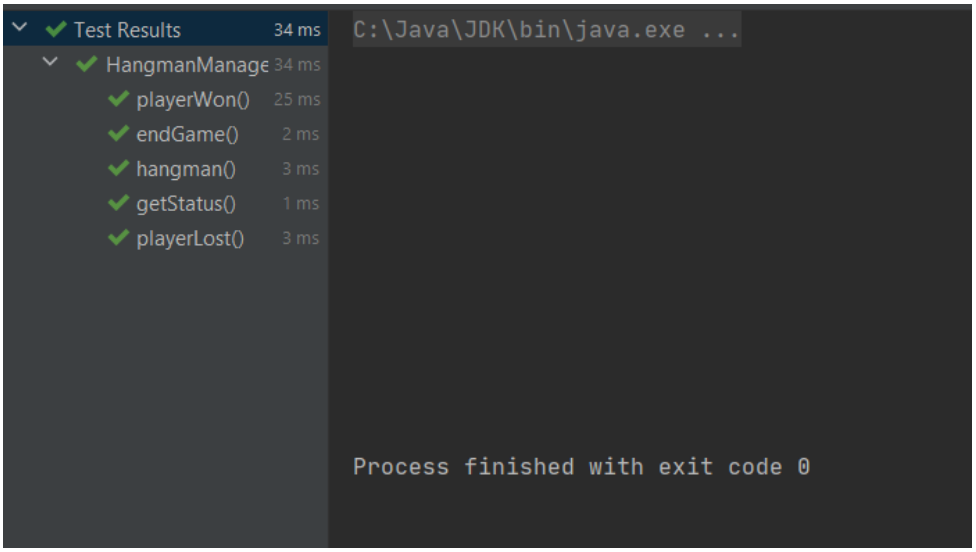
Test getStatus():

```
@org.junit.jupiter.api.Test
void getStatus() {
    HangmanManager h = new HangmanManager();
    h.setAnswer("you");
    ArrayList<Character> word = new ArrayList<~>();
    word.add('y');
    word.add('o');
    word.add('u');
    h.setCorrectGuess(word);
    assertEquals( expected: "YOU SUCCEED", h.getStatus());

    HangmanManager ha = new HangmanManager();
    ha.setAnswer("what");
    ArrayList<Character> w = new ArrayList<~>();
    w.add('y');
    w.add('o');
    w.add('u');
    w.add('y');
    w.add('o');
    w.add('u');
    w.add('y');
    w.add('o');
    w.add('o');
    w.add('u');
    ha.setWrongGuess(w);
    assertEquals( expected: "YOU DIED", ha.getStatus());
}
```

Test hangman():

```
@org.junit.jupiter.api.Test
void hangman() {
    HangmanManager h = new HangmanManager();
    h.setAnswer("name");
    ArrayList<Character> word = new ArrayList<>( initialCapacity: 4);
    word.add('w');
    word.add('q');
    word.add('t');
    word.add('u');

    String expected = "\t|----------\n" +
            "\t|\t\t|\n" +
            "\t|\n\t|\n\t|\n\t|\n\t|\n" +
            "____|____";
    h.setWrongGuess(word);
    assertEquals(expected, h.hangman());
}
```

Automated Tests:



Source Code : https://gitlab.lnu.se/1dv600/student/xz222bb/assignment-3

Reflection:

Through this iteration, I understand better about testing. The process and different concepts of testing. Also, I realized it is important to know what to test and how to test. Testing is related to the code that is implemented, so the results of testing help me to develop code more efficiently.
